



UNIVERSIDAD  
AUTÓNOMA DE  
AGUASCALIENTES

# PROYECTO FINAL

*Sistemas Operativos*

SIMULADOR DE TÉCNICAS DE COLOCACIÓN DE  
MEMORIA Y PLANIFICACIÓN DE PROCESOS BASADO EN  
BUDDY SYSTEM Y ROUND ROBIN.

## INTEGRANTES:

Mariana Isabel Alvarado de la Huerta  
Danna Cristina Castro Hernández  
Darely Quezada Guerrero  
Celia Fernanda Vela Uribe  
César Andrés Zuleta Malanco

## PROFESOR:

Francisco Guillermo Gutierrez Najera

# ÍNDICE

2

## **Introducción.**

Presentación del proyecto y sus objetivos.

3

## **Contenido.**

Explicación de cómo funciona el algoritmo (división de memoria en bloques, combinación de bloques al liberar, fragmentación). Descripción de cómo se asignan los turnos, el manejo del "cuanto" y cómo se elige el siguiente proceso.

8

## **Implementación Buddy System, conceptual y código.**

Explicación del concepto y código de los algoritmos utilizados.

11

## **Implementación Round Robin, conceptual y código.**

Explicación del concepto y código de los algoritmos utilizados.

13

## **Conclusiones.**

Reflexión grupal e individual sobre los aprendizajes, retos y logros del proyecto.

16

## **Bibliografía.**

Lista de libros, artículos y recursos electrónicos consultados.

18

## **Anexo A.**

Instrucciones detalladas para compilar y ejecutar el proyecto. Mención de dependencias, comandos y pasos necesarios.

# INTRODUCCIÓN.

En la actualidad, la gestión eficiente de los recursos en un sistema operativo es esencial para garantizar un buen rendimiento en las aplicaciones y sistemas computacionales.

Los simuladores de memoria, procesos y recursos se han convertido en herramientas clave en la enseñanza y comprensión de estos conceptos. Permiten a estudiantes, investigadores y profesionales analizar el comportamiento de sistemas complejos, sin necesidad de acceder directamente a hardware físico o plataformas operativas reales. Estos simuladores ofrecen un mejor enfoque para experimentar con diferentes técnicas y algoritmos, proporcionando una comprensión más profunda de su funcionamiento.

Este documento detalla el desarrollo, diseño y funcionalidad de un programa diseñado para la simulación y manejo de memoria y recursos en un sistema operativo. Esta herramienta se centra en implementar dos técnicas importantes para los sistemas operativos: el Buddy System para la gestión de memoria y el Round Robin para la planificación de procesos. Ambas técnicas son pilares fundamentales en la administración de sistemas operativos modernos debido a su eficacia y versatilidad en diferentes contextos.

El objetivo principal del programa es proporcionar una herramienta didáctica que permita a los usuarios simular, visualizar y entender cómo interactúan los procesos, cómo se realiza la asignación de memoria y cómo se gestionan los recursos en un sistema operativo. En esta simulación, los usuarios pueden observar el impacto de diferentes configuraciones en el rendimiento del sistema, así como analizar estadísticas reales que les ayuden a evaluar la eficiencia de las técnicas implementadas.

Este proyecto no solo busca simular el uso de un sistema operativo, sino también relacionar los conceptos de la gestión de sistemas operativos y su aplicación práctica de simulación. Con este proyecto se espera que los usuarios puedan adquirir habilidades y conocimientos para diseñar, implementar y optimizar sistemas operativos en diversos escenarios.

# CONTENIDO

En el diseño de sistemas operativos, la asignación de memoria del núcleo es un aspecto crítico que involucra la asignación de memoria para operaciones y estructuras de datos a nivel del núcleo. Cuando un proceso se ejecuta en modo de usuario y solicita memoria adicional, el núcleo mantiene la asignación de páginas de la lista de marcos de página libres. Por lo tanto, la asignación de memoria del núcleo se puede definir de la siguiente manera:

"El proceso mediante el cual el núcleo del sistema operativo asigna memoria para sus operaciones internas y estructuras de datos se denomina asignación de memoria del núcleo".

En el sistema operativo, la asignación de memoria del kernel es una tarea crítica, por lo tanto, debe realizarse de forma correcta y eficiente.

Dependiendo de los requisitos del sistema y del tipo de asignación de memoria, existen dos técnicas importantes de asignación de memoria del núcleo, a saber, el Buddy System y el sistema de bloques.

La técnica de Buddy System en términos más generales es una técnica eficiente de administración de memoria dinámica que divide la memoria disponible en bloques de tamaño que son potencias de dos. Este método es ampliamente utilizado debido a su capacidad para manejar fragmentación interna y externa de manera efectiva. Es un algoritmo de asignación de memoria que funciona dividiendo la memoria en bloques de un tamaño fijo. Cada bloque de memoria en este sistema tiene un "orden".

De esta forma, cuando se realiza una solicitud de memoria, el algoritmo encuentra el bloque de memoria más pequeño disponible (que sea suficiente para satisfacer la solicitud). Si el bloque es más grande que el tamaño solicitado, se divide en dos bloques más pequeños de igual tamaño (también conocidos como "buddies"). Uno de ellos se marca como libre y el segundo como asignado. Luego, el algoritmo continúa recursivamente hasta que encuentra el tamaño exacto de la memoria solicitada o un bloque que sea el tamaño más pequeño posible.

El Buddy System reduce la fragmentación al garantizar que los bloques asignados estén siempre contiguos y alineados. Esto significa que no hay huecos ni agujeros entre los bloques, y que los bloques comienzan y terminan en direcciones fijas. Además, el Buddy System también reduce la fragmentación externa, que se produce cuando hay suficiente memoria libre pero no en un solo bloque contiguo. El Buddy System evita esto fusionando buddies libres adyacentes en bloques más grandes cuando se desasignan. De esta manera, el algoritmo mantiene una jerarquía de bloques libres de diferentes tamaños y puede encontrar rápidamente el que mejor se adapte a cualquier solicitud.

Algunas de sus ventajas son:

- **Velocidad y facilidad de implementación**, ya que solo requiere operaciones bit a bit y estructuras de datos simples.
- **Flexible y adaptable**, capaz de asignar bloques de diferentes tamaños según la demanda.
- **Eficiente y compacto**, lo que minimiza la fragmentación y el desperdicio de espacio en la memoria.
- **Escalable y robusto**, capaz de manejar solicitudes de memoria grandes y dinámicas.
- El Buddy System **utiliza una estructura de árbol binario** para representar bloques de memoria usados o no utilizados.

El Buddy System también tiene algunas desventajas que limitan su rendimiento:

- **La fragmentación interna** cuando el bloque asignado es mayor que el tamaño solicitado, limitan su rendimiento.
- **La sensibilidad al orden y el tamaño de las solicitudes** de asignación y desasignación también limitan el rendimiento de este sistema y un mayor riesgo de errores y corrupción debido a la dependencia del cálculo y el mantenimiento correctos de las direcciones y tamaños de los amigos.
- Puede provocar desperdicio innecesario.

El Buddy System se usa en sistemas donde la eficiencia y el tiempo son críticos, como en sistemas operativos de tiempo real o gestión de memoria de kernel. Sin embargo, su efectividad depende del patrón de solicitudes de memoria de las aplicaciones.

Por otro lado, una de las principales responsabilidades de un sistema operativo es gestionar la ejecución de múltiples tareas o procesos en una sola CPU. Para ello, el sistema operativo utiliza un algoritmo de programación que decide qué tarea ejecutar a continuación y durante cuánto tiempo. La técnica de Round Robin es un algoritmo de planificación de procesos ampliamente utilizado en sistemas operativos, especialmente en entornos de tiempo compartido.

Su objetivo principal es asignar de manera equitativa el tiempo de CPU entre todos los procesos listos para ejecutarse. Este enfoque garantiza que todas las tareas obtienen una parte equitativa de los recursos del sistema y evita que una tarea monopolice la CPU por tiempo indefinido. Las tareas se almacenan en una cola circular y el algoritmo elige la primera tarea de la cola, la ejecuta para un cuanto y la mueve al final de la cola. El algoritmo repite este proceso hasta que se completan o finalizan todas las tareas. El algoritmo round-robin es adecuado para sistemas que necesitan manejar muchas tareas cortas e interactivas con prioridades similares.

El algoritmo de Round Robin se basa principalmente en dos factores:

- **Quantum de tiempo:** Se define un intervalo de tiempo fijo, conocido como quantum o slice, durante el cual cada proceso puede ejecutarse. Este quantum debe ser lo suficientemente largo para que la sobrecarga de los cambios de contexto no afecte significativamente el rendimiento del sistema.
- **Cola de procesos:** Los procesos listos para ejecutarse se organizan en una cola circular. El planificador selecciona el primer proceso de la cola, le asigna la CPU durante un quantum y, si el proceso no termina en ese tiempo, lo coloca al final de la cola. Este ciclo se repite continuamente.

Este algoritmo se implementa utilizando una estructura de datos conocida como cola circular, la cual facilita el ciclo continuo de los procesos. La cola circular asegura que todos los procesos tengan la oportunidad de ejecutarse en un orden justo, lo cual es fundamental en la planificación round robin para evitar que algún proceso monopolice el tiempo de la CPU y garantizar así una distribución equilibrada de los recursos del sistema.

Para comprender mejor cómo funciona la planificación de Round Robin, consideremos un ejemplo. Supongamos que tenemos tres procesos: A, B y C, con los siguientes tiempos de ejecución: A (5 unidades de tiempo), B (3 unidades de tiempo) y C (2 unidades de tiempo).

Proceso	Tiempo de Ejecución
A	5
B	3
C	2

Utilizando un quantum de tiempo de 2 unidades, la tabla a continuación muestra cómo se asignan los recursos del sistema a cada proceso en cada intervalo de tiempo:

Quantum	Proceso en Ejecución
1-2	A
3-4	A
5-6	B
7-8	B
9-10	C
11-12	A
13-14	B
15-16	B
17-18	C
19-20	A

En este ejemplo, podemos observar cómo los procesos A, B y C se ejecutan de manera secuencial, asignándoles a cada uno un quantum de tiempo de 2 unidades. Cuando un proceso no se completa dentro de su quantum asignado, se pasa al siguiente proceso en la cola y se le dará otra oportunidad más adelante.

Las principales ventajas del algoritmo de Round Robin son:

- Garantiza una **respuesta rápida** para los procesos interactivos.
- Al asignar quantum de tiempo a cada proceso, se **evita que un proceso se ejecute indefinidamente**, lo que mejora la equidad y la eficiencia en el uso de los recursos del sistema.
- **Fácil de implementar y entender**, ya que solo requiere una simple cola y un temporizador.
- **Justo y equitativo**, ya que le da a cada tarea una parte igual del tiempo de CPU y evita la inanición.
- **Receptivo**, ya que reduce el tiempo medio de espera y el tiempo de respuesta de las tareas.
- El tiempo de respuesta es el tiempo que tarda una tarea desde su llegada hasta su finalización.

Si bien Round Robin es una técnica efectiva para distribuir los recursos del sistema, también puede presentar algunas desventajas. Por ejemplo:

- **Cambios de contexto frecuentes**, que son las operaciones que guardan y restauran el estado de una tarea cuando sale y entra en la CPU. Son costosos en términos de tiempo y recursos, ya que implican cambiar la memoria, los registros y otros componentes de la CPU.
- **Alta varianza en el tiempo de respuesta**, que es el tiempo que tarda una tarea desde su llegada hasta su primera ejecución en la CPU.
- **Rendimiento bajo**, que es el número de tareas completadas por unidad de tiempo.
- **Degradación en su rendimiento** y una posible demora en su finalización.

La planificación de Round Robin se utiliza comúnmente en sistemas operativos y entornos de multiprocesamiento, donde múltiples procesos compiten por los recursos del sistema. También se utiliza en sistemas de tiempo compartido, donde varios usuarios acceden simultáneamente a un sistema central.



# IMPLEMENTACIÓN: BUDDY SYSTEM

```
class BuddySystem {
    final int totalMemory = 1024; // Total memory in KB
    final int minBlockSize = 32; // Minimum block size: 32 KB
    final Map<int, int> memoryUsage = <int, int>{}; // Tracks the memory usage per block size

    BuddySystem() {
        // Initialize memory blocks
        int blockSize = totalMemory;
        while (blockSize >= minBlockSize) {
            memoryUsage[blockSize] = totalMemory ~/ blockSize;
            blockSize ~/= 2;
        }
    }

    void debugMemoryBlocks() {
        print(object: "Memory blocks status:");
        memoryUsage.forEach(action: (int blockSize, int availableBlocks) {
            print(object: "Size: $blockSize KB, Free: $availableBlocks");
        });
        print(object: "\n");
    }

    int getAvailableMemory() {
        int availableMemory = 0;
        memoryUsage.forEach(action: (int blockSize, int availableBlocks) {
            availableMemory += blockSize * availableBlocks;
        });
        return availableMemory;
    }

    int? allocateMemory(int requestSize) {
        int blockSize = _findBlockSize(requestSize);
        if (blockSize == 0) {
            print(object: "Not enough memory for the requested size: $requestSize KB");
            return null;
        }

        // Allocate a block
        if (memoryUsage[blockSize]! > 0) {
            memoryUsage[blockSize] = memoryUsage[blockSize]! - 1;
            print(object: "Allocated $blockSize KB for request of $requestSize KB");
            return blockSize;
        }

        // Split larger blocks
        if (_splitBlock(targetBlockSize: blockSize)) {
            memoryUsage[blockSize] = memoryUsage[blockSize]! - 1;
            print(object: "Allocated $blockSize KB for request of $requestSize KB after splitting");
            return blockSize;
        }

        print(object: "No available memory blocks for size $blockSize KB");
        return null;
    }

    int _findBlockSize(int requestSize) {
        int blockSize = minBlockSize;
        while (blockSize < requestSize && blockSize <= totalMemory) {
            blockSize *= 2;
        }
        return blockSize > totalMemory ? 0 : blockSize;
    }

    bool _splitBlock(int targetBlockSize) {
        int largerBlockSize = targetBlockSize * 2;
        if (memoryUsage[largerBlockSize] != null && memoryUsage[largerBlockSize]! > 0) {
            memoryUsage[largerBlockSize] = memoryUsage[largerBlockSize]! - 1;
            memoryUsage[targetBlockSize] = memoryUsage[targetBlockSize]! + 2;
            print(object: "Split $largerBlockSize KB into two $targetBlockSize KB blocks");
            return true;
        }
    }
}
```

```

void freeMemory(int blockSize) {
    if (!memoryUsage.containsKey(key: blockSize)) {
        print(object: "Invalid block size: $blockSize KB");
        return;
    }

    memoryUsage[blockSize] = memoryUsage[blockSize]! - 1;
    print(object: "Freed $blockSize KB block");
    _mergeBlocks(blockSize: blockSize);
}

void _mergeBlocks(int blockSize) {
    while (blockSize < totalMemory) {
        if (memoryUsage[blockSize]! >= 2) {
            memoryUsage[blockSize] = memoryUsage[blockSize]! - 2;
            int largerBlockSize = blockSize * 2;
            memoryUsage[largerBlockSize] = (memoryUsage[largerBlockSize] ?? 0) + 1;
            print(object: "Merged two $blockSize KB blocks into one $largerBlockSize KB block");
        } else {
            break;
        }
    }

    debugMemoryBlocks();
}

```

Se implementó la técnica de Buddy System, un esquema de gestión de memoria que divide la memoria en bloques de tamaños que son potencias de 2, permitiendo una asignación y liberación eficiente. En este código:

- **totalMemory**: Define la cantidad total de memoria disponible (1 MB = 1024 KB).
- **minBlockSize**: Representa el tamaño más pequeño de bloque que se puede manejar.
- **memoryUsage**: Es un mapa que rastrea el número de bloques disponibles por tamaño.
- Inicializa los bloques de memoria desde minBlockSize hasta totalMemory. Cada bloque se registra en memoryUsage con el número de bloques disponibles.
- **debugMemoryBlocks()**: Muestra el estado actual de los bloques de memoria: tamaño de bloque y bloques libres.
- **getAvailableMemory()**: Calcula la memoria disponible sumando todos los bloques libres.
- **int? allocateMemory(int requestSize)** y **int \_findBlockSize(int requestSize)**: Se encargan de la asignación de memoria y la búsqueda de tamaño del bloque adecuado. También se encuentra el tamaño de bloque mínimo que puede satisfacer la solicitud. Si la solicitud excede totalMemory, devuelve 0.
- Si hay bloques disponibles del tamaño adecuado, se asigna uno y se reduce el conteo en memoryUsage.
- En **\_splitBlock**: Si no hay bloques disponibles, intenta dividir un bloque más grande para crear dos bloques más pequeños.
- **bool \_splitBlock(int targetBlockSize)**: Divide un bloque más grande en dos bloques más pequeños del tamaño requerido.

- **freeMemory(int blockSize):** Libera un bloque de memoria que habia aumentado el conteo en memoryUsage e intenta fusionar bloques del mismo tamaño para formar bloques más grandes en \_mergeBlocks.
- Finalmente, **\_mergeBlocks(int blockSize):** Verifica si hay al menos dos bloques libres del mismo tamaño y si es así, los fusiona en un bloque más grande y actualiza memoryUsage.

# IMPLEMENTACIÓN: ROUND ROBIN

```
class RoundRobinScheduling {
    Timer? _schedulingTimer;

    void startScheduling(List<ProcessID> processes, Function updateCallback) {
        const Duration quantum = Duration(seconds: 1);
        _schedulingTimer = Timer.periodic(duration: quantum, callback: (Timer timer) {
            if (processes.isEmpty) return;

            ProcessID process = processes.removeAt(index: 0);

            if (process.remainTime != null && process.remainTime! > 0) {
                process.status = ProcessStatus.running;
                updateCallback();

                process.remainTime = process.remainTime! - process.quantumTime!;

                if (process.remainTime! > 0) {
                    process.status = ProcessStatus.ready;
                    processes.add(value: process);
                } else {
                    process.status = ProcessStatus.block;
                }
            }

            updateCallback();
        });
    }

    void stopScheduling() {
        _schedulingTimer?.cancel();
    }
}
```

En esta parte del algoritmo se utiliza la técnica de Round Robin en la que divide el tiempo de CPU equitativamente entre todos los procesos. Esto asegura que ningún proceso acapare la CPU, logrando un sistema más justo y eficiente, además de usar un temporizador para simular la ejecución en tiempo real, ajustando los tiempos de ejecución de los procesos según el quantum y actualizando la interfaz o el estado del simulador periódicamente.

En este código:

- La variable llamada **\_schedulingTimer** es un temporizador que controla la ejecución de las operaciones de planificación de procesos. En este caso se utiliza para ejecutar el algoritmo en intervalos definidos por quantum.
- El método **startScheduling(List<ProcessID> processes, Function updateCallback)** inicia el proceso de planificación Round Robin.
- Y el método **stopScheduling()** Detiene el temporizador que controla la planificación.

Durante el método `startScheduling`:

- Se define el quantum, el intervalo de tiempo durante el cual se le asigna la CPU a cada proceso. Se establece en 1 segundo.
- Se crea un temporizador que ejecutará el bloque de código cada segundo (según el quantum).
- Si la lista de procesos está vacía, la planificación se detiene ya que no hay nada que ejecutar.
- Se selecciona el proceso al frente de la cola y se elimina de la lista para ser procesado.
- Durante la ejecución del proceso:
  - Se comprueba que el proceso tiene tiempo restante.
  - Se cambia el estado del proceso a `running`.
  - El tiempo restante del proceso se reduce según el quantum.
  - Si el proceso aún tiene tiempo restante, se vuelve a insertar en la cola con estado `ready`.
  - Si no tiene tiempo restante, se marca como `block`, lo que significa que se ha terminado su ejecución.
- Cada vez que se ejecuta un cambio en el estado o la lista de procesos, se llama al `updateCallback` para actualizar.
- Finalmente el método `stopScheduling` detiene el temporizador y la ejecución del método de Round Robin.

# CONCLUSIÓN

En general, ambos conceptos, Buddy System y Round Robin, son fundamentales en el diseño y funcionamiento de sistemas operativos, aunque tratan y mejoran problemas diferentes. Sin embargo, comparten el objetivo común de maximizar la eficiencia y equidad en el uso de los recursos del sistema.

Ambos conceptos optimizan los recursos, hacen más simple su implementación en distintos sistemas utilizando estructuras de datos simples y se comprometen a ser más eficientes y equitativos.

Sin embargo, y a pesar de que comparten algunos comportamientos, sus objetivos y aplicaciones son completamente distintos. El Buddy System se centra en la administración eficiente de la memoria, resolviendo problemas como la fragmentación y la reutilización de bloques. A diferencia del el Round Robin que se ocupa de la planificación de procesos, asegurando tiempos de respuesta justos y predecibles para todos los procesos en ejecución.

Además, los desafíos que enfrentan también son diferentes. Mientras que el Buddy System debe lidiar con la fragmentación interna y la pérdida de memoria potencial, el Round Robin enfrenta el reto de ajustar el quantum de tiempo para evitar tanto la sobrecarga como la ineficiencia.

El Buddy System muestra cómo la memoria puede dividirse y combinarse dinámicamente para minimizar la fragmentación, mientras que el Round Robin demuestra cómo los recursos de CPU pueden distribuirse de manera equitativa entre procesos, garantizando tiempos de respuesta consistentes. Sin embargo, la efectividad de ambos algoritmos depende de una configuración cuidadosa: el tamaño de los bloques en el Buddy System y el quantum en el Round Robin son parámetros críticos que deben ajustarse según las características del sistema y las aplicaciones.

En resumen, tanto el Buddy System como el Round Robin ejemplifican la importancia de diseñar algoritmos que combinen eficiencia, equidad y simplicidad. Esto no solo resuelven problemas específicos de gestión de recursos, sino que también destacan entre los diferentes componentes de un sistema operativo, donde la memoria y la CPU trabajan en conjunto para proporcionar un sistema funcional, confiable y eficiente para las aplicaciones y los usuarios.

# CONCLUSIONES

## ISABEL:

Existen dos opciones para el diseño de sistemas operativos, Buddy System y Round Robin, las cuales ayudan a la codificación de estos al optimizar el manejo de memoria dentro del programa, lo cuál mejora la eficiencia de los sistemas. Ambos algoritmos cuentan con ventajas y desventajas, las cuales se tienen que tomar en cuenta al decidir cual de los dos algoritmos usar.

## DANNA:

Buddy System es un algoritmo de gestión de memoria que se basa en la división de la memoria en tamaños fijos en potencia de 2. Su uso brinda eficiencia en la asignación y liberación de memoria debido a que permite combinar y dividir bloques de manera rápida y sencilla, reduciendo la fragmentación externa y facilitando la reutilización de los recursos disponibles.

Round Robin es un algoritmo de planificación de procesos en sistemas operativos que asigna tiempo de CPU a cada proceso en partes iguales y en un orden cíclico. Su uso brinda equidad y visualización debido a que asegura que todos los procesos tengan acceso a la CPU de manera regular, evitando el monopolio de recursos por parte de un único proceso y garantizando tiempos de respuesta consistentes en sistemas multitarea

## DARELY:

En este documento se vieron las diferencias entre Buddy System y Round Robin y se revisó cómo se implementan en un código real y su manejo de memoria dentro de este, y cómo funciona con el manejo de datos reales, así como la teoría de estos algoritmos. El uso del sistema creado sirve para ejemplificar estos algoritmos al simular procesos y manejo de recursos.

# CONCLUSIONES

## **FER:**

El manejo de memoria cuenta con varias opciones de algoritmos para usarse en código. Con investigación de cada una (en este caso Buddy System y Round Robin) y una codificación correcta resulta más eficiente manipular la memoria y recursos con procesos. En este proyecto se aprecia la diferencia entre Buddy System y Round Robin así como su aplicación en un caso real.

## **CÉSAR:**

El manejo de memoria es fundamental para los sistemas operativos. Aprender sobre algoritmos como Buddy System y Round Robin permite apreciar cómo estos manejan la creación y manipulación de memoria. Además, la creación de un código de ejemplo, aunque sencillo, da una noción clara de cómo se utilizan y cómo funcionan estos algoritmos en sistemas completos. Conocer sobre Buddy System y Round Robin fue conocimiento clave para el control de recursos y procesos.



# BIBLIOGRAFÍA

Boutnaru, S., PhD. (2023, 29 julio). Operating Systems – Buddy Memory Allocation - Shlomi Boutnaru, Ph.D. - medium. Medium. <https://medium.com/@boutnaru/operating-systems-buddy-memory-allocation-fcbb476eefe4>

Casero, A. (2024, 18 marzo). La lógica Round Robin en programación. KeepCoding Bootcamps. <https://keepcoding.io/blog/la-logica-round-robin-en-programacion/>

GeeksforGeeks. (2023, 14 abril). Allocating kernel memory (buddy system and slab system). GeeksforGeeks. <https://www.geeksforgeeks.org/operating-system-allocating-kernel-memory-buddy-system-slab-system/>

¿Cómo mejora el sistema de compañeros la asignación de memoria? (2023, 31 de octubre). LinkedIn: inicio de sesión o registro. <https://es.linkedin.com/advice/3/how-does-buddy-system-improve-memory-allocation-esq5f?lang=es>

Sistemas Operativos – Memoria (página 2). (s.f.). Monografias.com. <https://www.monografias.com/docs113/sistemas-operativos-memoria/sistemas-operativos-memoria2>

<https://es.linux-console.net/?p=23368>

Digital, T. (2023, 7 de julio). Planificación de Round Robin: Definición y Ejemplos. Informática y Tecnología Digital. <https://informatecdigital.com/algoritmos/planificacion-de-round-robin-definicion-y-ejemplos/>

Okyere, J. (2024, 10 de febrero). ¿Cómo programa el algoritmo round-robin las tareas en los sistemas operativos? LinkedIn: inicio de sesión o registro. <https://es.linkedin.com/advice/0/how-does-round-robin-algorithm-schedule-tasks-irh4c?lang=es&lang=es>

# BIBLIOGRAFÍA

<https://www.fing.edu.uy/inco/cursos/sistoper/recursosTeoricos/6-SO-Teo-Planificacion.pdf?>

Broks, N. (2024, 12 de agosto). Algoritmo de programación Round Robin con ejemplo. Guru99. <https://www.guru99.com/es/round-robin-scheduling-example.html>

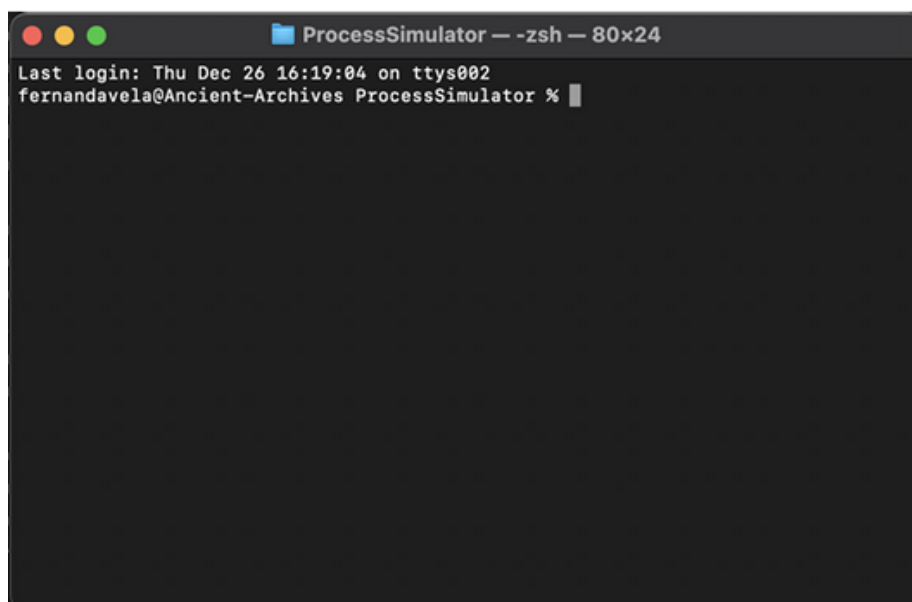
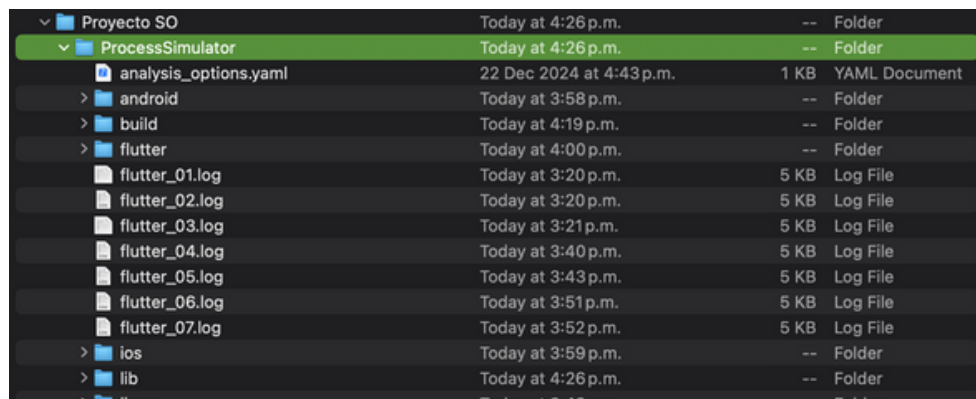
ouchouchbaby. (2015, 18 de marzo). Round Robin Algorithm Tutorial (CPU Scheduling) [Video]. YouTube. <https://www.youtube.com/watch?v=aWlQYlIBZDs>

Sudhakar Atchala. (2019, 17 de octubre). Allocating Kernel Memory in operating system | buddy system memory allocation|slab allocation in os [Video]. YouTube. <https://www.youtube.com/watch?v=sKWSk0o-TxE>

# ANEXO A

## MANUAL DEL USO.

Abra la carpeta zip o descargue el programa desde el git que compartió. Cuando termine de procesarse abra la terminal o Windows PowerShell en la raíz del proyecto: ProcessSimulator.



Verifica que tu extensión de Flutter esté completamente operativa. Puedes hacerlo ejecutando el comando `flutter doctor -v`. También puedes evitar problemas ejecutando `flutter clean` o `flutter pub cache repair`.

Una vez que hayas confirmado que Flutter está activo y funcionando, puedes iniciar el proyecto ejecutando `flutter run`.

```
ProcessSimulator — dart --packages=/Users/fernandavela/flutter/flutter/...
[fernandavela@Ancient-Archives ProcessSimulator % flutter run
Connected devices:
macOS (desktop) • macos • darwin-arm64 • macOS
14.4.1 23E224 darwin-arm64
Mac Designed for iPad (desktop) • mac-designed-for-ipad • darwin • macOS
14.4.1 23E224 darwin-arm64
Chrome (web) • chrome • web-javascript •
Google Chrome 131.0.6778.205

No wireless devices were found.

[1]: macOS (macos)
[2]: Mac Designed for iPad (mac-designed-for-ipad)
[3]: Chrome (chrome)
Please choose one (or "q" to quit):
```

Despues seleccione la tercera opción (3) que iniciará la aplicación en Chrome.

```
ProcessSimulator — dart --packages=/Users/fernandav...
[fernandavela@Ancient-Archives ProcessSimulator % flutter run
Connected devices:
macOS (desktop) • macos • darwin-arm64 • macOS
14.4.1 23E224 darwin-arm64
Mac Designed for iPad (desktop) • mac-designed-for-ipad • darwin • macOS
14.4.1 23E224 darwin-arm64
Chrome (web) • chrome • web-javascript •
Google Chrome 131.0.6778.205

No wireless devices were found.

[1]: macOS (macos)
[2]: Mac Designed for iPad (mac-designed-for-ipad)
[3]: Chrome (chrome)
Please choose one (or "q" to quit): 3
Launching lib/main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
```

Si la aplicación es correcta y todo funciona sin problemas deberías ver esto:

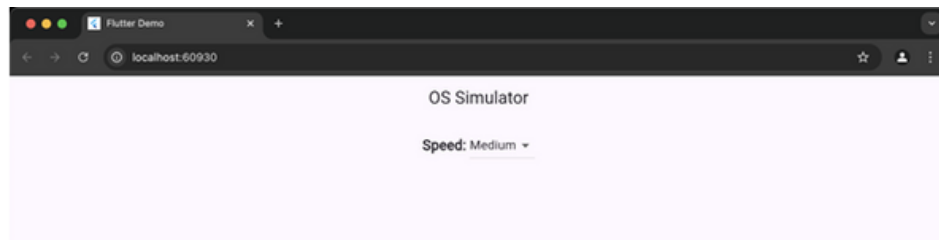
```
[3]: Chrome (chrome)
Please choose one (or "q" to quit): 3
Launching lib/main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome... 9.2s
This app is linked to the debug service: ws://127.0.0.1:60611/MGTuIZMC2zE=/ws
Debug service listening on ws://127.0.0.1:60611/MGTuIZMC2zE=/ws

🔥 To hot restart changes while running, press "r" or "R".
For a more detailed help message, press "h". To quit, press "q".

A Dart VM Service on Chrome is available at: http://127.0.0.1:60611/MGTuIZMC2zE=
Memory blocks status:
Size: 1024 KB, Free: 1
Size: 512 KB, Free: 2
Size: 256 KB, Free: 4
Size: 128 KB, Free: 8
Size: 64 KB, Free: 16
Size: 32 KB, Free: 32

The Flutter DevTools debugger and profiler on Chrome is available at:
http://127.0.0.1:9101?uri=http://127.0.0.1:60611/MGTuIZMC2zE=
```

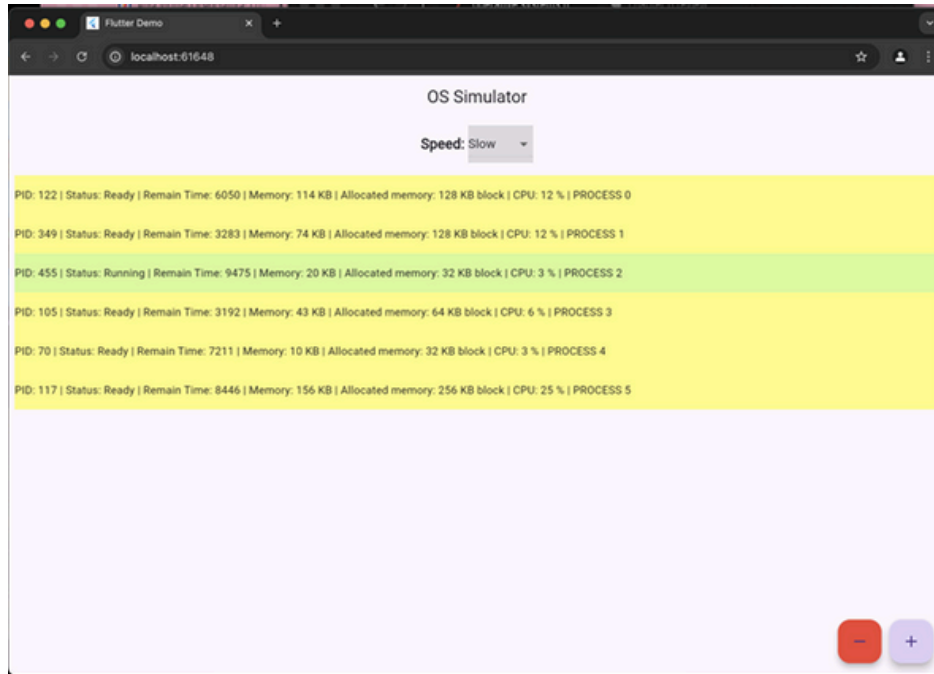
Esto también abrirá el navegador web Chrome de la aplicación. Las tres cosas principales que debes tener en cuenta son la marcación rápida, botón agregar y el botón detener. El **marcado rápido** está ubicado en la **parte superior central de la pantalla** y ayudará a **modificar la velocidad a la que aparecen los procesos** cuando se agregan.



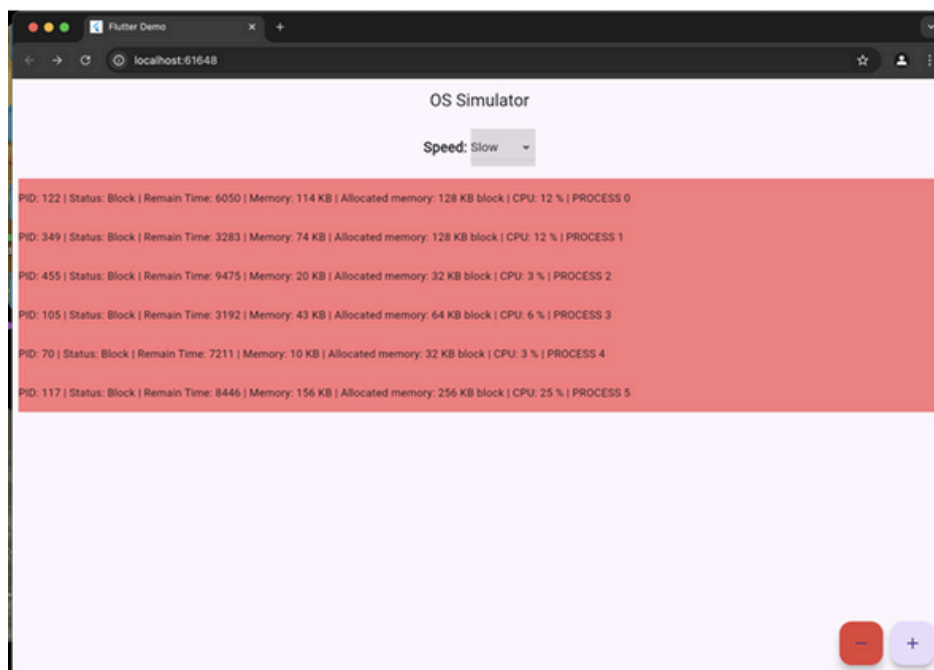
Los dos botones están ubicados en la **esquina inferior derecha**. El **botón Agregar** agregará los procesos con aleatorización memoria y tiempo. El **botón de parar** bloqueará todos los procesos creando una detención total.



Use el botón de agregar para añadir los procesos como desee. Los procesos que no se estén ejecutando estarán en un estado de listo. Estos procesos tendrán un fondo amarillo. Los procesos en ejecución cambiarán a color verde.

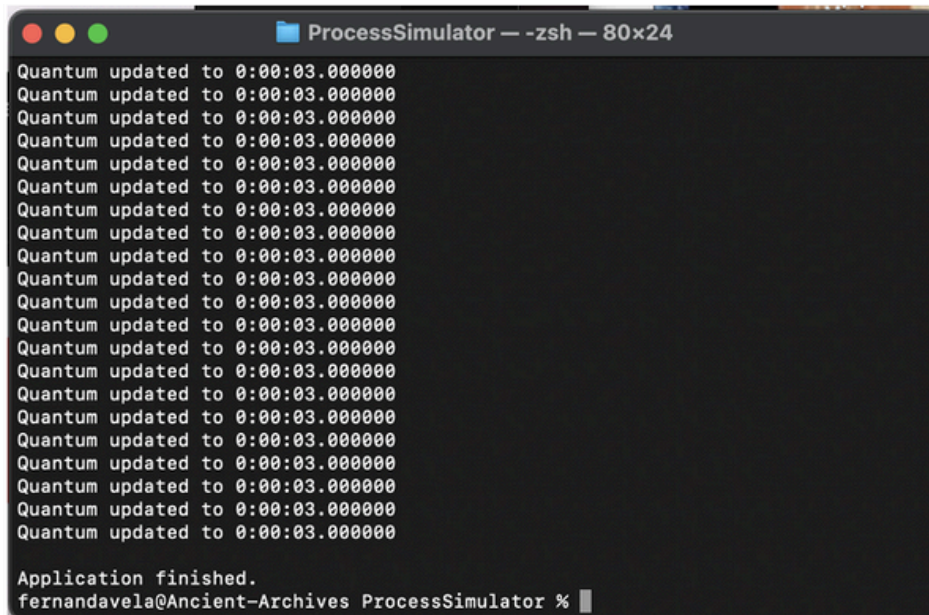


Cuando el botón de detener bloquee todos los procesos y genere una detención total, esto cambiará el color de los procesos a rojo.





Para salir del programa, regresa al terminal y simplemente escribe una "q". Esto cerrará la aplicación y también la ventana web de Chrome.

A terminal window titled "ProcessSimulator — -zsh — 80x24" with standard macOS window controls. The terminal displays a series of 20 lines, each reading "Quantum updated to 0:00:03.000000". After the 20th line, the text "Application finished." appears. The prompt "fernandavela@Ancient-Archives ProcessSimulator %" is visible at the bottom with a cursor.

```
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Quantum updated to 0:00:03.000000
Application finished.
fernandavela@Ancient-Archives ProcessSimulator %
```